TACUE: A Timing-Aware Cuts Enumeration Algorithm for Parallel Synthesis

Mahmoud Elbayoumi, Mihir Choudhury†, Victor Kravets†, Andrew Sullivan†, Michael Hsiao
Mustafa Elnainay‡
ECE Dept., Virginia Tech, Blacksburg, VA, USA, 24061
†IBM T. J. Watson Research Center, Yorktown Heights, NY, USA, 10598
‡Computer and Systems Engineering Department, Alexandria University, Alexandria, Egypt, 21544
mbayoumi@vt.edu,{choudhury, kravets, sullia}@us.ibm.com
hsiao@vt.edu, ymustafa@alexu.edu.eg

ABSTRACT

Achieving timing-closure has become one of the hardest tasks in logic synthesis due to the required stringent timing constraints in very large circuit designs. In this paper, we propose a novel synthesis paradigm to achieve timing-closure called *Timing-Aware CUt Enumeration (TACUE)*. In TACUE, optimization is conducted through three aspects: (1) a new divide-and-conquer strategy is proposed that generates multiple sub-cuts on the critical parts of the circuit; (2) two cut enumeration strategies are proposed; (3) an efficient parallel synthesis framework is offered to reduce computation time. Experiments on large and difficult industrial benchmarks show the promise of the proposed method.

Categories and Subject Descriptors

B.6 [**Electronic Design Automation**]: Logic Synthesis

Keywords

Timing Closure, BDD bidecomposition, parallel synthesis

1. INTRODUCTION

Logic synthesis is the automated generation of an optimized logic networks (in terms of delay [1], area and/or power [2]) from another unoptimized/sub-optimized logic network. Over the years, many researchers have been interested in optimizing certain dedicated hardware components. For example, Sklyarov *et al.* [3] had proposed a novel technique for synthesizing parallel hierarchical finite state machines; Roy *et al.* [4] had proposed a customizable prefix graph structures that yield adders with optimal performance-area trade-off. Other researchers had developed various synthesis algorithms for dedicated platforms. For example, Uma *et al.* [5] had explored constraint synthesis optimization technique for targeted FPGA device. Another research aims toward building general methodologies that is capable of efficiently synthesizing both control and data-path components [6].

DAC'14 , June 01 - 05 2014, San Francisco, CA, USA

http://dx.doi.org/10.1145/2593069.2593227

This paper fits in the last group, that is, general timing-aware synthesis. As far as we know, there are two major bodies of work that are related to delay-optimization with cuts generation. The first was proposed by Baneres *et al.* [7], in which a dominator-based partitioning technique is used to find topologically ordered clusters in the circuit-under-optimization (CUO), followed by logic restructuring on these clusters. The second timing-aware work conducts cuts enumeration on *And-Inverter-Graphs (AIGs)*. For example, Chatterjee *et al.* [8] had proposed a cut factorization scheme to enumerate bounded size cuts up to 16 inputs. Their technique is usually used in technology mapping and re-writing. Martinello *et al.* [9] had extended the concept of factor cuts to KL-cuts, where K is number of inputs and L is number of outputs in a circuit cut. Because factor cuts are not restricted to convex cuts [10], an unbounded amount of area increase may result. This increase in area would lead, in many cases, to undesirable degradation in circuit timing behavior. Although KL-cuts extend the application of factor cuts to peep-hole optimization and regularity extraction, they still suffer from having a restricted number of inputs and work exclusively with *AIGs*.

Our work is uniquely different from the previous work in several points. Unlike to Baneres work [7] that only groups nodes in the critical paths and generates a single solution, we enumerate sub-cuts in dominator-based partitions. As a result, we are less likely to be stuck at local optima as our approach explores more possible solutions to improve the timing behavior of CUO. Secondly, their approach allows the grouping of different dominant cuts, which can result in a significant increase of area, thereby indirectly degrade the timing performance of the optimized solution. On the contrary, we propose two different cutting strategies, one that aims to preserve the topological structure of CUO and the other one would allow for other possible topologies.

In contrast with work by Chatterjee [8] on factor cuts, our approach could handle larger cuts (TACUE have been tested on up to 60 inputs sub-cuts). In addition, our technique runs on a general circuit graph. Thus, each vertex represents a general Boolean function (not just "AND" function as in AIG). Subsequently, our method can be applied in all synthesis stages and it is not restricted only to technology mapping stage.

Previous experiments were conducted on fairly small circuits (ISCAS'85, ISCAS'89, ITC'99 and some other small circuits). In our case, we conducted experiments on very large industrial benchmarks. In addition, previous methods only optimized their circuits using no longer actively developed SIS tool [11]. In contrast, we first apply extensive optimization techniques (i.e. BooleDozer [12] industrial synthesis tools, ABC [13], SIS, etc.) before using our synthesis framework to show that our synthesis framework exhibits a superior outcome for very-large very-hard-to-optimize circuit instances. Finally, we propose an efficient parallel synthesis framework for applying different synthesis optimization techniques in the generated TACUE sub-cuts.

TACUE shows up to 22.22% reduction in the number of levels compared with state-of-the-art timing-aware synthesis algorithms. With adequate tuning, we could achieve up to 45.72% of reduction in the worst slack with a slight increase in area of 1.87% in average (6.51% in the worst case). With our work-balanced parallel synthesis engine, we get a speed up of 2.18×, 3.99×, 5.42×, and 7.12× over 2, 4, 6 and 10 processes, respectively.

The rest of the paper is organized as follows. The next section briefly introduces the preliminaries on Binary Decision Diagram (BDD) bi-decomposition, time-driven logic bi-decomposition, vertex dominator and dominant cuts. TACUE algorithm is presented in Section 3. In Section 4, we apply TACUE algorithm to different cutting approaches: *topology-aware* cuts and *topology-masking* cuts. We propose an efficient parallel synthesis framework for TACUE cuts in Section 5. Our results are presented in Section 6. Finally, the paper is concluded in Section 7.

2. PRELIMINARIES

2.1 BDD Bi-decomposition

An efficient BDD bi-decomposition is proposed in [14]. It starts by building the BDD of each output. Then, it recursively decomposes each output BDD to two smaller logically related BDDs. In order to achieve a provably optimum variable partition for the given logic function, we use a fast, scalable algorithm proposed in [15]. It constructs an undirected graph called the Blocking Edge Graph (BEG). Please refer to [15] (and reference therein) for more details.

2.2 Time-Driven Logic Bi-Decomposition

Time-driven logic bi-decomposition is proposed in [16]. It synthesizes a timing-aware circuit by first bi-decomposing the Boolean representation of the circuit, then it re-balances the functions using a tree-height reduction technique. Kravets *et al.* [17] had proposed a general symbolic decomposition template for logic synthesis to infer its decomposition patterns. Using this template, the decomposition is performed in a Boolean domain unrestricted by the representation of a function. Bi-decomposition technique in [16] is applied iteratively on the decomposed templates.

2.3 Vertex Dominator and Dominant Cuts

Vertex dominator and dominant cuts have been used in many EDA domains [18]. Due to lack of space, we refer readers to [18] for more details.

3. TIMING-AWARE CUT ENUMERATION

```
1: TACUE(C)
2: c_base = createBaseCone(C)
3: queue.enq(c_base)
4: while queue.size() > 0 do
5:    v = queue.deque()
6:    combCount = CalCombCount(v.boundary())
7:    for i = 1 to combCount do
8:       nPairEnum(v, i, queue)
9:    end for
10: end while
```

Figure 1: TACUE algorithm outlines.

```
1: nPairEnum(v, i, queue)
2: c = getCritVertex(v)
3: if i = 1 then
4:    for j = 1 to c.length() do
5:       cnew = createNewCut(v, c[i])
6:       queue.enq(cnew)
7:    end for
8: else
9:    for j = 1 to c.length() do
10:      vnew = createNewCut(v, c[i])
11:      nPairEnum(vnew, i − 1, queue)
12:   end for
13: end if
```

Figure 2: Enumerates all n combinations at the boundary of a sub-cut.

We target to generate time-critical sub-cuts from bigger cuts. In other words, given a cut in the CUO, our objective is to enumerate, heuristically, sub-cuts in the critical paths of the CUO. These sub-cuts will be passed later to various logic synthesis optimization techniques. If they successfully find better solutions, a heuristic is used to select the optimal choice among them. Then, the optimal choice will be admitted, and hence, this would contribute to the overall timing closure of the CUO.

The TACUE algorithm for enumerating critical sub-cuts is shown in Fig. 1. TACUE starts with a critical cut C. Then, it uses a Breadth-first (BF) approach for enumerating the sub-cuts. First, TACUE creates base sub-cuts, which contain the root with all direct children by calling *createBaseCut* (line 2). TACUE adds the base cut to a queue (line 3). Then it loops until the queue becomes empty (line 4-10). It dequeues a sub-cut from the queue (line 5) and calculates the possible combinations that could be conducted on the boundary of this sub-cut (line 6). Finally, it enumerates all combinations using *nPairEnum* function (line 7-9).

Fig. 2 depicts the algorithm for *nPairEnum* function. *nPairEnum* enumerates all i combinations at the boundary of a sub-cut v and adds it to *queue*. We measure the criticality of a vertex by *vertex slack* (the difference between the required arrival time and the actual arrival time). *nPairEnum* enumerates only the critical vertices, that is, it has a cut-off value on vertex slack. Thus, vertices with slack larger than certain threshold will not be added to the enumerated sub-cuts list.

Fig. 3 illustrates our approach in which TACUE generates the time-critical sub-cuts. Let the slack cut-off value be 1. TACUE takes the original cut t_O (the cut that needs to be enumerated as depicted in Fig. 3.a) and a set of tun-

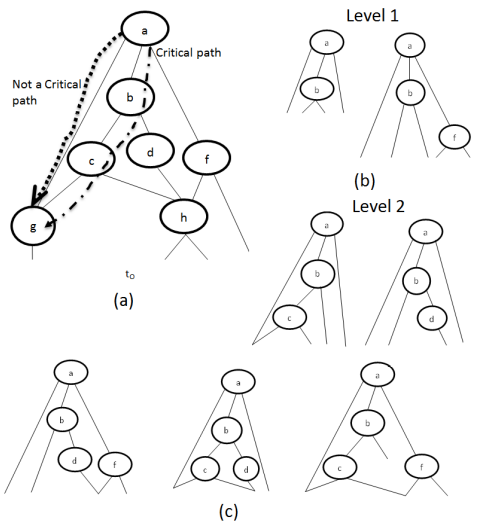

Figure 3: Cut enumeration illustration. (a) the original cut, (b) sub-cuts generated in the first level, and (c) sub-cuts generated in the second level.

ing parameters. The cut-enumeration algorithm starts with the root a of t_O and enumerates sub-cuts in a BF manner. The vertices of cut t_O are labeled with nodes $a, b, \ldots, etc.$ The direct children of a are g, b and f. Without loss of generality, let the slack of vertex g be 3 (which is greater than the cut-off slack value). Then, vertex g and all of its children will not be included in any enumerated sub-cuts. Now, TACUE will enumerate all possible combinations for the other direct children nodes b and f. As vertex b has slack value of zero (b is in a critical path, and all critical vertices have a zero slack) and is a direct child of a, it will be included in any enumerated sub-cuts. For node f, let us assume that its slack is 1, thus it is not on the critical path. However, its slack is smaller than the cut-off value, thus, it will be included in the future sub-cuts. However, because it is not on a critical path, it does not need to be included in every sub-cut. Thus, at this level, we have two sub-cuts $(a, \{b\})$ and $(a, \{b, f\})$ as depicted in Fig. 3.b.

In the second round, the cut-enumeration algorithm will enumerate cuts in the next level for all cuts in the first level as depicted in Fig. 3.c. It starts with cut $(a, \{b\})$, the direct children of this cut are c, d and f. We have 7 combinations, The children vertices will be enumerated one by one. The resultant sub-cuts will be $(a, \{b, c\})$, $(a, \{b, d\})$ and $(a, \{b, f\})$. The next step would be enumerating them two by two. The resultant sub-cuts will be $(a, \{b, c, d\})$, $(a, \{b, c, f\})$ and $(a, \{b, d, f\})$. Finally we consider three nodes at a time. Thus, all three direct children will be taken altogether in sub-cut $(a, \{b, c, d, f\})$. This cut is not shown in the Fig. 3 because the cut-off limit for generated sub-cuts is 8 (which is a user input). In the case that the cut-off limit is increased, this final cut and sub-cuts in deeper levels will be generated and added.

4. CUT ENUMERATION

In the previous section, we described how TACUE takes a cut C and enumerates timing-critical sub-cuts from C. However, we did not describe how cuts were generated in TACUE. In this section, we describe two divide-and-conquer strategies to generate these cuts.

```
1: GenerateStructAwareCuts()
2: GenDomCuts()
3: CL = FilterCuts()
4: for all C in CL do
5:    TACUE(C)
6: end for
```

Figure 4: Topology-Aware cuts generation algorithm.

```
1: GenerateStructMaskCuts()
2: CL = GetCritOutCuts()
3: repeat
4:    for all C in CL do
5:       TACUE(C)
6:    end for
7:    CL =getNextCritCuts()
8: until CL is NOT empty
```

Figure 5: Topology-Masking cuts generation algorithm.

4.1 Topology-Aware Cuts

In some cases, we need to preserve the connectivity of the CUO. This is useful when we start with an initially "'good" topology, and it is desirable to keep the same topology to be used later.

One way to achieve connectivity preservation is by restricting changes to be made only inside each dominant cut. This intuition is motivated by the self-contained nature of dominant cuts, that is, any vertex in a dominant cut does not fanout to any vertex outside that cut. In other words, we model each dominant cut as a single super node and we restrict the change of logic to occur only inside these super nodes. This restriction results in preserving the general connectivity of circuit nearly the same.

Fig. 4 depicts *topology-aware* cuts generation algorithm. It starts by generating dominant cuts (line 2). Then the critical dominant cuts are enumerated (line 3). Finally, it iterates on all filtered cuts and enumerates them (lines 4-5).

4.2 Topology-Masking Cuts

Fig. 5 depicts the *topology-masking* cuts generation algorithm. Contrary to the *topology-aware* cutting strategy, this strategy does not take the CUO connectivity preservation into consideration because we may start with an initially poor timing-performance CUO or locally optimized CUO. The algorithm starts with identifying critical outputs of the CUO and generates cuts from these outputs (line 2). These cuts have the critical sink vertices as a root and all its fan-in source vertices as the boundary vertices. Secondly, it enumerates sub-cuts from these critical cuts (lines 4-5). If a sub-cut is being accepted as a new solution, it is committed to CUO. The aforementioned process is repeated at the critical boundary vertices of the new committed sub-cut (line 7).

5. PARALLEL SYNTHESIS FRAMEWORK

In this section, the optimized parallel strategies for TACUE in a synthesis framework are described.

1: ParrSynth()
2: CL = EnumerateCuts()
3: **for all** C in CL **do**
4: ParSynth(C)
5: **end for**

Figure 6: Parallel Synthesis Algorithm

Table 1: Circuit Statistics

ckt	IN	OUT	DFF	GATES	LV	AR	WS
ia0	279	624	968	12339	27	48602	-36.335
ia1	229	517	870	146608	26	47110	-28.619
ia2	204	508	840	127823	26	39667	-19.344
ib0	253	626	963	149566	26	48234	-37.644
ib1	201	515	867	146530	27	46763	-26.999
ib2	176	505	841	127782	27	39126	-18.384

5.1 Parallel Framework - Optimized Approach

In order to have a well-balanced parallel framework, we propose to split the cut enumeration step from parallel synthesis. This is based on noting that cut enumeration only takes a small fraction of time compared to the synthesis step. Thus, we will not have a tangible performance degradation if we enumerate the cuts sequentially. Meanwhile, we boost the performance of the major part of our framework by having a full parallelism in the synthesis optimization stage.

Fig. 6 lists our proposed parallel framework algorithm. As shown in Fig. 6, sub-cuts are first pre-computed. Next, they are evenly distributed among different workers (which achieves a well-balanced workload). Thirdly, each worker applies different synthesis optimization techniques on sub-cuts assigned to it. All successful sub-cuts are sent back to the master process, and the master determines which sub-cut would be committed to the original circuit. The used criteria aim to decrease the number of levels while maintaining a limited percentage of area increase.

6. RESULTS

The proposed TACUE algorithm and parallel synthesis framework have been developed with C++ and the performance was evaluated on 11 dedicated 2.7 GHz Intel Xeon cores, running a 64-bit Linux distribution. We have compiled our program with g++ under -O3 option. We have used large industrial benchmarks to evaluate our work. The characteristics for our benchmarks are reported in Table 1. We applied many optimizations before applying TACUE (i.e., SIS, ABC, BooleDozer and industrial synthesis tools ...etc.). In doing so, we guarantee that our benchmarks are already "'well optimized", making further optimization harder. For each circuit, the number of inputs (IN) is first listed, followed by the number of outputs (OUT), the number of sequential elements (DFF), the total number of gates (GATES), the number of levels (LV), the total area (AR) for the combinational part of the benchmarks (measured in number of basic unit cells). Finally, the worst slack (measured in picoseconds) is reported in the last column. Note that all of the circuits have a negative worst slack, which mean that, **the current state of art synthesis tools could not achieve timing-closure on these designs.**

6.1 Topology-Aware Cuts

Table 2: No. of Level Reduction with Topology-Aware Cuts

ckt	Itr. # 1		Itr. # 2		MLRP (%)
	BD	TD	BD	TD	
ia0	25	24	25	23	14.81
ia1	25	25	24	23	11.53
ia2	25	24	23	23	11.53
ib0	25	24	24	23	11.53
ib1	25	24	25	23	14.81
ib2	25	24	24	23	14.81

Table 3: Area Report for Topology-Aware Cuts

ckt	Itr. # 1		Itr. # 2		MAIP (%)
	BD	TD	BD	TD	
ia0	48698	48630	48668	48616	0.1975
ia1	47148	47213	47174	47231	0.2569
ia2	39656	39676	39666	39713	0.1160
ib0	48304	48223	48302	48195	0.1451
ib1	46771	46899	46782	46962	0.4256
ib2	39216	39312	39227	39275	0.475

Table 2 reports the number of levels after applying TACUE using the *topology-aware* cuts. We apply TACUE for 2 iterations (Itr. # 1 and Itr. # 2) with BDD bi-decomposition (BD) and time-driven logic synthesis (TD). The last column reports the *Maximum Level Reduction Percentage (MLRP)* for each case. The results showed that we can reduce the number of levels by 14.81% for *topology-aware* cuts. Table 3 reports the corresponding circuit area. The last column reports *Maximum Area Increase Percentage (MAIP)* for each case. Our approach showed that TACUE has a very slight area increase of only 0.475%.

Table 4 reports the number of dominant cut (DC) computed and the number of accepted dominant cuts (AC). AC is defined as the number of dominant cuts that the synthesis algorithm had successfully reduced its number of levels. We report results for 3 iterations. For each iteration, we ran both BDD Bi-decomposition and time-driven logic bi-decomposition. We always had a higher acceptance rate in TD case over BD because TD tends to better optimize the cut in terms of delay. In addition, the number of AC decreases with the increasing number of iterations because the *topology-aware* cutting strategy restricts TACUE to preserve the topology. Thus, we do not have a large room for changing the design structure.

6.2 Topology-Masking Cuts

Table 5 reports the number of levels after applying TACUE with *topology-masking* cuts. TACUE was applied for 3 iterations for both BDD BD and TD as before. The results show that we could get up to 22.22% reduction (compare with 14.81% for *topology-masking* cuts in Table 2) in the number of levels. This is due to that we did not require TACUE to preserve the topology.

Table 6 reports the percentage area increase from the topology-masking technique. We noticed that TD perform poorly (especially with the increase in number of iterations) from the area point of view (up to 44.7% area increase). This is because we allow for changing the topology, and TD had a large acceptance rate. On the other hand, BD synthesis

Table 4: Dominant Cut Statistics for Topology-Aware Cuts

ckt	Itr. # 1				Itr. # 2				Itr. # 3			
	BD		TD		BD		TD		BD		TD	
	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC
ia0	710	60	710	149	790	14	812	60	784	3	817	24
ia1	653	56	653	144	720	14	772	56	719	1	771	20
ia2	634	35	634	118	640	7	669	45	641	2	709	18
ib0	675	61	675	161	721	8	773	65	745	3	766	29
ib1	593	55	593	136	628	16	688	70	652	8	717	28
ib2	617	41	617	131	630	8	674	48	636	2	683	17

Table 5: No. of Level Reduction with Topology-Masking Cuts

ckt	Itr. # 1		Itr. # 2		Itr. # 3		MLRP (%)
	BD	TD	BD	TD	BD	TD	
ia0	26	24	23	22	23	21	22.22
ia1	25	23	23	21	23	21	19.23
ia2	25	23	23	21	23	21	19.23
ib0	24	23	23	21	23	21	19.23
ib1	25	24	23	21	22	21	22.22
ib2	25	24	23	22	22	21	22.22

Table 6: Area Report for Topology-Masking Cuts

ckt	Itr. # 1		Itr. # 2		Itr. # 3		MAIP
	BD	TD	BD	TD	BD	TD	
ia0	48889	49658	50829	60527	51773	66322	36.5
ia1	47753	52981	48127	61422	49415	67599	43.5
ia2	39702	39757	40113	48385	40447	54049	36.3
ib0	49299	54703	51004	62566	51376	68565	42.15
ib1	47267	50691	49022	58715	50326	67698	44.7
ib2	39315	39352	40891	48683	41957	52999	35.5

would reduce number of levels (up to 22%) while maintaining an adequate area increase (7.21% in circuit ib2). This increase in area will be reflected on the physical synthesis stages as it will be discussed later. We also noticed that, if we restrict the number of iteration in this stage to one, we would get enhancement on both logic and physical synthesis stages. This is because the number of levels is highly reduced from the first iteration with a limited percentage of area increase.

6.3 Impacts of TACUE on Physical Synthesis

We have run our physical synthesis tool on the circuits optimized by TACUE for *topology-aware* cuts. We report the worst slack in Table 7. The *Maximum Worst Slack Increase Percentage (MWSIP)* is reported in the last column. We could gain 21.16% increase of worst slack on average and 45.72% in the best case.

The impact of TACUE on physical synthesis is also investigated. Table 8 reports the worst slack after each iteration. Results show that we could gain 18.54% increase in the worst slack on average and 31.23% in the best case.

We also manually conduct experiments with different tuning parameters to optimize the overall flow of our synthesis tool. Table 9 depicts the best results we could obtain for each benchmark. We report worst slack and area in each benchmark for base case (BC) and best case (BSC). In addition, we report the synthesis technique (ST) we had utilize in the best case. We have 3 cases in which BD is superior to TD and other 3 cases in which TD is superior to BD. Moreover, we report the cutting technique (CT) we had used (TA: for *topology-aware* and TM: for *topology-masking*). The results show that TM is generally superior to TA. The Worst Slack Reduction percentage (WSRP) show that we could obtain an average 21.98% and up to 45.72% in the best case (for ia2). Results on *Level Reduction Percentage (LRP)* show that we have an average of 9.37% LRP with minimum of 3.84% (in ia1) and 14.81% (in ia0). Results for *Area Increase Percentage* (AIP in the last column) show that we have an average of 1.869% increase in area with minimum increase of 0.2569% (in ia2) and maximum area increase of 6.52% (in ia0).

Table 8: Worst-Slack Report for Topology-Masking Cuts

ckt	Itr. # 1		Itr. # 2	Itr. # 3	Itr. # 4	MWSIP
	BD	TD	BD	BD	BD	
ia0	-35.5	-31.3	-30.5	-28.4	-30.7	21.9
ia1	-26.3	-36.5	-30.0	-31.5	-26.4	8.13
ia2	-18.0	–	-19.2	-13.3	-16.1	31.2
ib0	-32.2	-40.1	-35.1	-36.5	-35.8	14.49
ib1	-25.4	-29.5	-23.9	-24.2	-27.6	11.64
ib2	-16.4	-14	-16.9	-19.5	-18.9	23.85

6.4 Parallel Synthesis Framework

Table 10 reports the time required for TACUE and our parallel Synthesis framework to optimize our benchmarks. The time is reported for one iteration of *topology-aware* cuts and our synthesis method is time-driven logic bi-decomposition. We report time for 1, 2, 4, 6, 10 and 20 processes. We gain an average speed-up of 2.18×, 3.99×, 5.42×, 7.12× and 8.78× on 2, 4, 6, 10 and 20 processes, respectively. The speed-up showed that we could achieve a good work balance. The reason for the super-linear speed for 2 processors and almost linear for 4 processors is that these are conducted on quad-core processors. Thus, in case of 2 and 4 processes we get benefit from locality and advanced parallelism features on the same processor. Our results show that we still get noteworthy speed-up for 6 and 10 processes. This is due to the fact that TACUE takes a small fraction of the computation. In addition, the way we organize the parallelism framework is efficient.

7. CONCLUSION

In this paper, a novel paradigm to accomplish timing clo-

Table 7: Worst-Slack Report for Topology-Aware Cuts

ckt	Itr. # 1		Itr. # 2		Itr. # 3		Itr. # 4		MWSIP (%)
	BD	TD	BD	TD	BD	TD	BD	TD	
ia0	-32.602	-29.72	-31.342	-31.237	-33.355	-31.046	-28.579	-29.501	21.35
ia1	-28.213	-29.103	-26.842	-26.579	-28.333	-29.157	-26.648	-29.61	7.13
ia2	-15.803	-20.81	-14.583	-10.499	-13.019	-15.39	-15.395	-14.361	45.72
ib0	-34.172	-35.763	-42.46	-32.997	-34.076	-33.973	-33.525	-39.275	10.94
ib1	-24.881	-22.204	-22.861	-25.984	-26.114	-26.074	-25.387	-20.398	24.45
ib2	-17.331	-21.352	-15.19	-20.382	-15.81	-24.511	-17.126	-19.164	17.37

Table 9: Physical Synthesis Best Results Summary

ckt	WS		AREA		CT	ST	# of itr.	WSRP(%)	LRP(%)	AIP (%)
	BC	BSC	BC	BSC						
ia0	-36.335	-28.368	48602	51773	TM	BD	3	21.93	14.81	6.52
ia1	-28.619	-26.293	47110	47753	TM	BD	1	8.13	3.84	1.36
ia2	-19.344	-10.499	39667	39713	TA	TD	2	45.72	11.53	0.2569
ib0	-37.644	-32.189	48234	49299	TM	BD	1	14.49	3.84	2.21
ib1	-26.999	-22.204	46763	46899	TA	TD	1	17.76	11.11	0.2908
ib2	-18.384	-14	39126	39352	TM	TD	1	23.85	11.11	0.5776

Table 10: parallelism Results (time measured in seconds)

ckt/np.	1	2	4	6	10	20
ia0	1781.1	859.7	437.6	326.3	264.3	205.6
ia1	2015.8	935.1	516.0	351.6	303.3	227.6
ia2	1377.1	600.6	333.4	249.0	183.5	157.7
ib0	1735.3	903.7	497.6	368.1	280.8	208.3
ib1	1670.9	732.9	404.2	310.1	219.9	193.1
ib2	1588.7	658.2	374.2	277.1	197.1	168.1

sure of very large, previously optimized circuits is presented. In order to tackle the scalability problem in our industrial benchmarks, we propose a divide-and-conquer heuristic, which we call Time-Aware CUt Enumeration (TACUE) algorithm. The basic idea behind TACUE is to generate many well-chosen sub-cuts along the critical paths of the circuits. We apply different synthesis techniques to these sub-cuts, and we choose the best solution in terms of delay and area. Some circuits start with a "good" topology, while others do not have this feature. Thus, sometimes we need to make a decision whether to keep the current topology or not. Accordingly, two different cutting strategies have been proposed to handle this issue. Finally, we have also proposed an efficient parallel synthesis framework for TACUE. Significant reductions in worst slack was achieved with only a slight to moderate area overhead. Although TACUE, with synthesis framework, seems to be sequential in nature on first impression, we could come up with an elegant way to separate these data dependencies and sharing. The results show that we could gain almost linear and sometimes super-linear speedups.

8. REFERENCES

[1] T. Matsunaga et. al. Power and delay aware synthesis of multi-operand adders targeting lut-based fpgas. *sym. on Low-power elect. and des.*, 2011.

[2] W. Shiue. Power/area/delay aware fsm synthesis and optimization. *Elsevier Mic. J.*, 2005.

[3] V. Sklyarov et. al. Synthesis of parallel hierarchical finite state machines. *ICEE*, 2013.

[4] S. Roy et. al. Towards optimal performance-area trade-off in adders by synthesis of parallel prefix structures. *DAC*, 2013.

[5] R. Uma et. al. Performance enhancement through optimization in fpga synthesis: Constraint specific approach. 2013.

[6] L. Amarù et. al. Bds-maj: a bdd-based logic synthesis tool exploiting majority logic decomposition. *DAC*, 2013.

[7] D. Baneres et. al. Dominator-based partitioning for delay optimization. *Proc. of the 16th ACM G. L. sym. on VLSI*, 2006.

[8] S. Chatterjee et. al. Factor cuts. *ICCAD*, 2006.

[9] O. Martinello Jr et. al. Kl-cuts: a new approach for logic synthesis targeting multiple output blocks. *DATE*, 2010.

[10] R. Glantz et. al. Finding all convex cuts of a plane graph in cubic time. *Springer, Alg. and Comp.*, 2013.

[11] E. Sentovich et. al. Sis: A system for sequential circuit synthesis. *U. C.,B.*, 1992.

[12] L. Stok et. al. Booledozer: logic synthesis for asics. *IBM J. of R. & D.*, 1996.

[13] R. Brayton et. al. Abc: An academic industrial-strength verification tool. *CAV.*, 2010.

[14] C. Yang et. al. Bds: A bdd-based logic optimization system. *IEEE Trans. on CAD of IC and Sys*, 2002.

[15] M. Choudhury et. al. Bi-decomposition of large boolean functions using blocking edge graphs. *ICCAD*, 2010.

[16] J. Cortadella. Timing-driven logic bi-decomposition. *IEEE Trans. on CAD of IC and Sys*, 2003.

[17] V. Kravets et. al. Resynthesis of multi-level circuits under tight constraints using symbolic optimization. *ICCAD*, 2002.

[18] R. Tarjan. Finding dominators in directed graphs. *SIAM J. on Comp.*, 1974.